Washington State University Everett

AWS Data Science

Team Aang

Final Report

Russell Heppell, Madison Velasquez
Daniel Nuon, Alex McKee, Adam Clobes

Mentors: Bill Bonner, Geoff Allen

Professor: Bolong Zeng

Due 7 May 2021

Table of Contents

1. Executive Summary

This document encapsulates the entirety of the WSU Everett 2020-2021 AWS Cloud Sciences capstone. The capstone is an automated series of AWS services being used to streamline DOR and ESD data from a receiving portal to a front-facing website for researchers and general users to make queries on. The purpose of this project is to remove the cumbersome process that the data must go through currently into one that will automate monotonous steps and remove the time-consuming processes that are currently required to be done manually.

This document explores the background of the project, the issues that it is being made to alleviate, and the overall design description of the project and the motives behind each section. Within this description entails the variables of the data necessary for the researchers to properly perform their research and how those variables must be presented at the front end of the project. The high-level design of the project is illustrated to encompass the various requirements and standards needed for the project to be successful, and a description of how these requirements are broken down into more digestible sections.

These sections include the receival of the data, the processing of the data, and the delivery of the data. The receival of the data includes downloading the files from the secure email portal only accessible by registered WSU employees into a secure landing zone within AWS. The processing involves the encryption, cleaning, obfuscating, and hashing of the data as it is prepped to be uploaded. The delivery includes receiving the uploaded data inside a PostgreSQL database and aggregating it into meaningful visual information for the researchers and general users to view on a front-facing website.

These three sections further broken-down result in feasible tasks to be handled by different team members. These tasks include the receiving, security, obfuscation and cleaning, uploading, data storage, and creating the front-facing website.

This document concludes with the results of the design and the final version of the project, as well as the comparison between the original requirements and the final prototype. Within this document, the process of the creation of this project is detailed in the chronological nature of the individual tasks, with each section stepping through these tasks in order of which they occur in the full system’s use.

2. Introduction and Background

A team of researchers at WSU receive data quarterly from the Department of Revenue (DOR) and the Employment Security Department (ESD) to research and calculate Washington State business information. They receive 3 files from the ESD: ESD EQUI, ESD Tax, and Crosswalk, and 2 files from the DOR: DOR Qtr. Tax and DOR Reg Tax. With these 5 files, an obfuscation algorithm and a python script are used to hash and calculate a set of 7 variables, aggregated on year, region, and service type. The 7 variables are:

- Average Employment Data
- Closure Rate Data
- Cost of Labor
- Employment Hours Data
- Establishments
- Min Wage Data
- Revenue Data

One problem with their current process is that the data is stored on a physical drive at the Pullman campus, protected by various locked doors and keypad security systems, which creates a timely process for researchers to retrieve data. Another problem is that the python script that does the calculations and aggregations must be performed every time a researcher wishes to the view the data, which causes unnecessary data processing and extends the timeline even more for researchers wishing to view their data.

The goal of this project is to streamline this data research process, using various Amazon Web Services(AWS) to create a secure cloud-based web application with cloud data storage. Another crucial aspect of the project is to keep the security of the data consistent, which will be achieved with AWS services as well.

The solutions that the team chose and implemented over the course of this semester and last semester were done using the iterative design process, as well as concept generation to research and choose from various alternative solutions. The potential solutions were found using user stories and scenarios that were created from requirements given by the researchers and other stakeholders.

3. Project Design Description

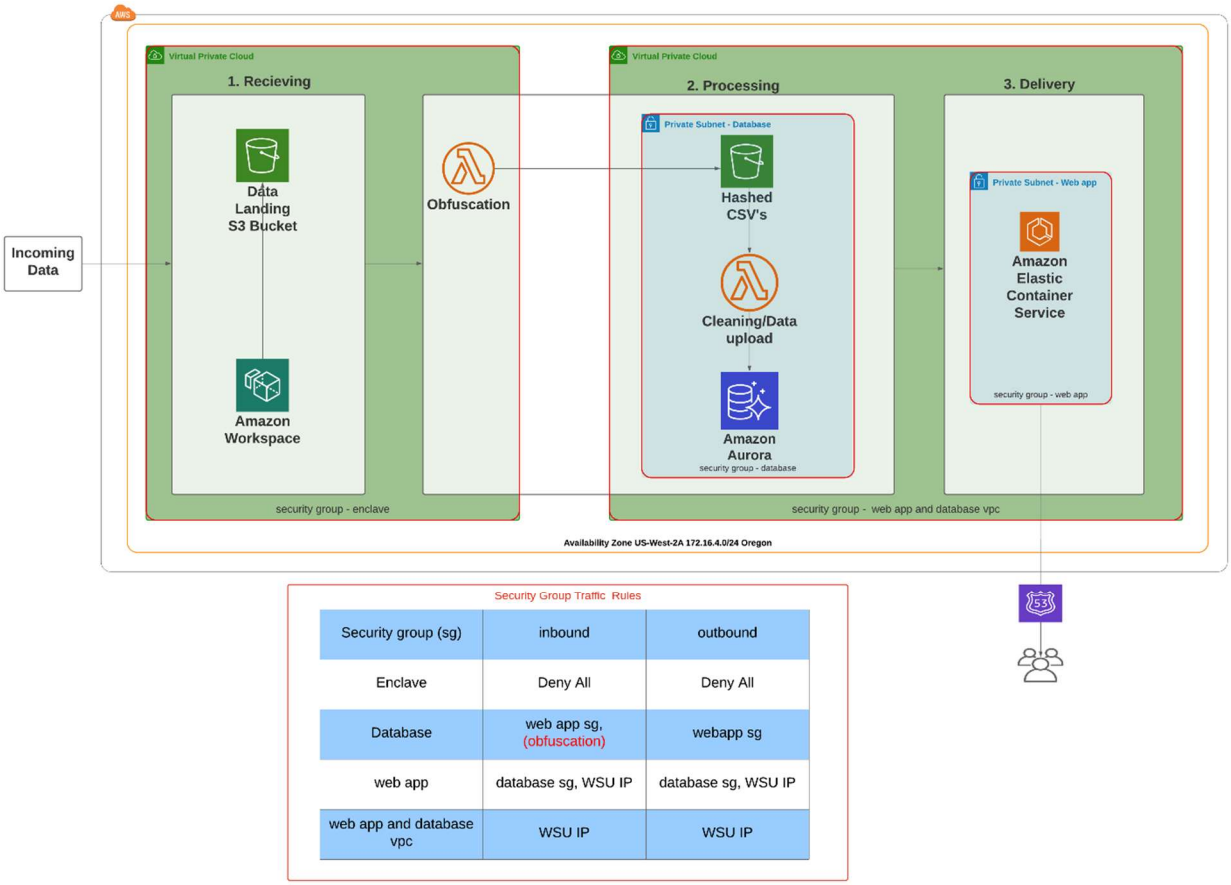


Figure 1. High level system design.

We divided the project into 3 main subsystems: Receiving, Processing, and Delivery. Figure 1 shows the high-level system design, encapsulated within AWS along with the security group rules. Below is a list of the AWS services utilized in the project, along with a short description of what they are (1).

Elastic Compute Cloud (EC2) – Virtual servers in the Cloud
Elastic Container Service (ECS) – Secure, reliable, scalable way to run containers in the Cloud
Lambda Functions – Run code in the cloud without servers
Relational Database Service (RDS) – Relational database service manager
S3 Bucket – Scalable storage in the Cloud
Virtual Private Cloud (VPC) – Provision an isolated section of the AWS cloud in a defined virtual network
WorkSpaces – Virtual desktops in the Cloud
CodeCommit – Cloud code repository for source control
Identity Access Management (IAM) – access management for AWS services

3.1. Receiving

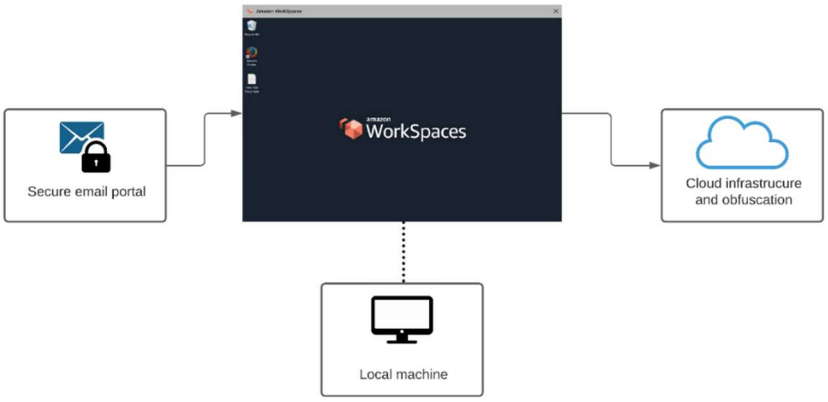


Figure 2. Workspace flow

Receiving the data from the secure email portal required the data to be highly protected during the most vulnerable interval of its processing. The solution to this was to create an Amazon Workspaces virtual machine instance to host the downloading from the portal into the AWS S3 bucket landing zone. By using Workspaces, the virtual machine completely insulates the data from the user's local machine as well as the outside internet, effectively creating a bubble around the raw data by downloading it while already inside AWS.

3.2. Processing

We defined our data processing to consist of four major tasks: data obfuscation, data cleaning, data upload, and data storage.

3.2.1. Obfuscation and Cleaning

The incoming files contain sensitive and or identifying information. This identifying and or sensitive information needed to be hashed to point where it was impossible to reverse engineer the original information. The project previously used a C# program and the SHA-256 algorithm from a Microsoft library, to hash out the files locally. One of the main goals of this project was to move the hashing and cleaning process to the Cloud and have it done in secure and automated way.

With these goals as a focal point, a few AWS services were relevant to our needs of hashing and cleaning; S3 buckets, IAM roles and lambdas. The S3 buckets are an AWS native service and inherently secure, the IAM roles allow the lambdas to get and put files from and to specified S3 buckets, and the lambdas are event driven functions that can run concurrently. The concurrent aspect is relevant if a large amount of files land in an S3 bucket at once then the obfuscation and cleaning will happen in parallel rather than serially.

The code for the obfuscation and cleaning is written in python and uses the Panda's library to work with the files. The Pandas library is a very powerful data-analysis extension to python and has many optimizations 'under the hood'. Additionally, there is good documentation from the Pandas project and extensive questions and answers on Stack Overflow.

Once a language was determined there were a few options of encryption algorithms to use, including MD5, SHA-128, SHA-256 and SHA-512. The previous encryption algorithm used was

SHA-256 and as it worked well previously, we continue to use this in the hashing process. It is quicker than SHA-512 and still effectively impossible to reverse engineer the original input (Gilbert H., Handschuh H. ,2004).

3.2.2. Upload

Uploading the data from the final S3 bucket into the database proved infeasible to do strictly via lambda functionality. While lambda functions are capable of cleaning and processing data sets as large as the ones this project calls for, they are not suited to handle the more complex and taxing process of uploading many rows of data into the context of a database's scaffolding. The approach to solve this was to offload the task of uploading the data from the lambda function onto another more apt platform or service that could then be configured to automatically handle the uploading from there. The lambda would then simply be triggered to upload the data to this outside platform.

This approach created an additional issue, however. With the lambda triggering its function every time a file lands in S3 bucket, multiple files being sent to this S3 asynchronously would, likewise, trigger multiple uploads into the outside platform. This would cause severe interferences within the platform and its automated uploading process. To avoid this, the lambda function was set on a scheduled timer to turn on and check the S3 for present files before pushing them to the platform.

3.2.3. Storage

In storing the data, the primary objective of this project was to utilize cloud-based storage using databases via AWS. Two databases were required to store obfuscated data and aggregated data. Within the obfuscated database, separate tables were created to store each individual file coming from the ESD and DOR as well as the crosswalk and key files. In addition, user login information was stored within this database. Within the aggregated data database, separate tables were created to store average employment, closure rate, cost of labor, employment hours, establishment, minimum wage, and revenue data.

When designing the schemas for the tables in the databases, we utilized previous output data that was provided by the researchers. These sample datasets were used to determine column names, data types, and aided in learning the output of the current data process. These datasets were also used in data upload testing.

3.3. Delivery

The delivery portion of the project was designed to prioritize four main things:

1. Security: Design secure login system with different types of users.
2. View ESD data : Design API to fetch data and design an organized way to see data with ajax data tables.
3. View DOR data: Design API to fetch data and design an organized way to see data with ajax data tables.
4. View aggregated data: Design filter system to view aggregated data from specific regions, service types, and years.

3.4 Security

The security of the system is a product of its design. Figure 1 above shows the high-level overview of the system, while the table gives more specifics on the security group rules. The security groups essentially act as firewalls, where resources in a group are limited to exchanging traffic only to those other security groups defined in the table. Additionally, subnets add another layer of security and further insulate the database and web application through network access control lists.

5. Project Management

Project Sections

This project had **four main sections**: creating the secure environment in AWS, receiving data, obfuscating data, and creating the website.

- Alex worked mainly on creating the environment in AWS and obfuscating the data with lambda functions.
- Adam worked on creating a secure method to receive the data.
- Madison worked on creating the website frontend and backend.
- Danny worked on designing and Implementing the database.
- Russell worked on integrating the database with the website and integrating the database into AWS RDS.

Team Meeting Schedule

The teams **meeting schedule** was facilitated via teams. The files were also managed on Teams in a Capstone teams' channel that included all members of the team and the mentors.

Team and Mentor Communication

Communication between the team and mentors was managed by the team's communication liaison, Russell Heppell. This communication happened with outlook email, and all members of the team were cc'd in these emails.

Communication within the Team

Internal communication within the team was managed with discord channel and an iMessage group chat.

The Gantt chart on the next page shows the timeline of how the above tasks were accomplished.

	Jan 19-22	Jan 25-29	Feb 1-5	Feb 8-12	Feb 15-19	Feb 22-26	Mar 1-5	Mar 8-12	Mar 15-19	Mar 22-26	Mar 29-Apr 2	Apr 5-9	Apr 12-16	Apr 19-23	Apr 26-30
Create Secure Environment for Data															
Create Each VPC and subnets with required security groups															
Connect hashCSV lambda from secure to database subnet in less secure VPC															
Place AWS Workspace in Secure VPC and configure user access to workspace															
Build AWS conformance pack															
Implement performance pack into AWS config, AWS Guard Duty, Security Hub, CloudWatch, CloudTrail															
Re-enforce External encryption with IAM Analyzer and VPC Flows															
Implement monitoring in Security Hub															
Recieve data from Secure Email Portal															
Create AWS WorkSpace and Format Server Role															
Create streamlined downloading interface within Workspace VM															
Obfuscate & Clean Data															
Obfuscate data															
Obfuscation Testing															
Clean Data															
Create Website Backend															
Automate S3-to-DB data uploading lambda-triggered Windows server															
Create database in PostgreSQL															
Import database into Amazon Aurora															
Write SQL for data variables and aggregation															
Create website backend using AWS API's															
Create Website Frontend															
Database Integration															
Login Authorization															
Website Query Linking															
Design GUI and researcher interface using Bootstrap															

Figure 3. Gantt Chart of this semester's tasks.

6. Results

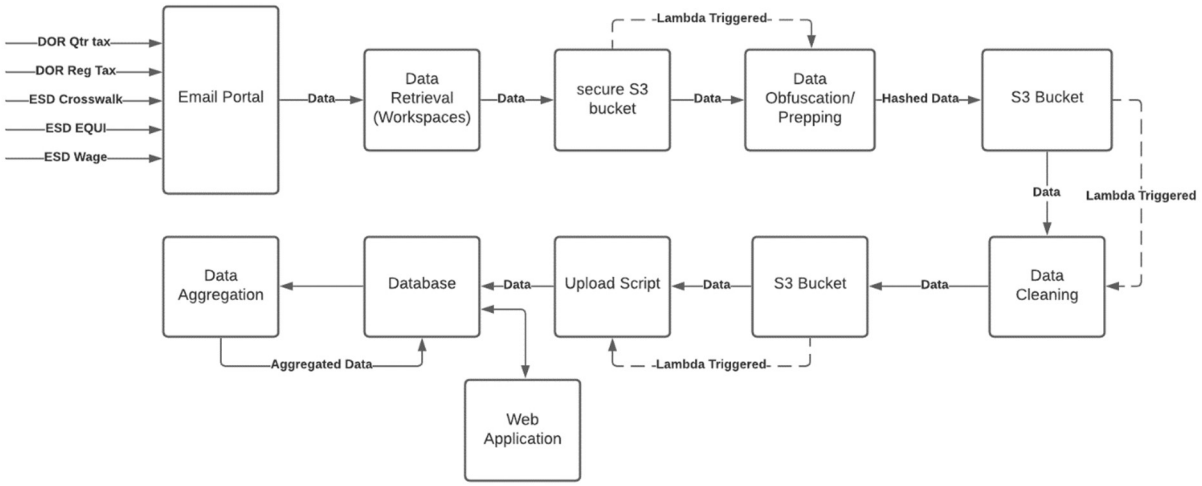


Figure 4. Data Flow Diagram.

Figure 4 shows the complete data flow from data receiving inside WorkSpaces, to data delivery in the web application. Lambda functions are used to trigger data movements from one process to another, with S3 buckets as landing zones between each lambda function.

A change in process that occurred this semester was the data aggregation process. Initially, the python script provided by the researchers was to be used in the data processing, but through mentor and team discussion, it was decided that the python script for performing aggregations was inefficient. The alternative is to perform the variable calculations and aggregations from inside the database and save that data to a second set of data tables. Doing the data calculations and aggregation from within the database is more secure, and PostgreSQL and SQL in general is more efficient for data aggregations for our purpose (2).

Table 1 below outlines the names of the S3 buckets and Lambda Functions created for the project. Table 2 in section 5.2 outlines each lambda function, its task, trigger, and where it sends data to.

Table 1. AWS S3 Buckets and Lambda Functions created for the project.

S3 Buckets	Lambda Functions
Input-asfoj33r	hashCSV
Output-21391i4	Cleaning-and-uploading
Output-cleaned-alex-2fi23o	Data-upload-ec2-start

6.1. Receiving

To receive the data, the Amazon Workspaces virtual machine is used to completely insulate the raw files from the outside world. The Workspaces instance was assigned a default IAM user that possessed an IAM role to grant it writing access to the S3 landing zone. While inside the Workspaces instance, the user simply downloads the raw data into a provided landing folder on the desktop. Once they are ready, the user then runs an included .bat file on the desktop to trigger the download. The .bat file runs several AWS console commands including a copy command to load the data into the S3 landing zone, then automatically and quietly deletes the raw data from the Workspaces folder. The first S3 in our data pipeline is input-asfoj33r, which starts the data processing discussed in 6.2.

6.2. Processing

Table 2. Lambda function tasks, triggers, and data destinations.

Lambda Function	Trigger	Task	Sends to
HashCSV	Data landing in input-asfoj33r	Hashes out sensitive data	Output-21391i4
Cleaning-and-uploading	Output-21391i4	Cleans data and formats for upload	Output-cleaned-alex-2fi23o
Data-upload-EC2-server	Runs every day at 3am	Turns on the server that runs the uploading script	Database

Table 2 above outlines the Lambda Functions along with their tasks, as well as the triggers set for them and the data destinations post-Lambda.

6.2.1. Obfuscation

The obfuscation happens in the hashCSV lambda function. The hashCSV function is triggered by the input-asfoj33r bucket, at which point it get the file from the bucket and starts to hash it. One of the requests for the project was to do quality assurance as the files come in from the agencies to ensure no frameshift has occurred, i.e., ensure there are no more or less columns in the data than expected. This QA check happens in the hashCSV lambda. See the timing benchmark for hashing in the appendix under the header ‘Lambda functions output’.

6.2.2. Cleaning

Each of the files that land in the output-21391i4 S3 bucket are the cleaned by the cleaning-and-uploading lambda. This lambda sorts by the file name and then calls the corresponding function to clean the file. Each file has its own quirks and as such each function is written differently. Once the file is cleaned it is sent to output-cleaned-alex-2fi23o bucket. This bucket is checked each night at 3 a.m., if there are files in it, they are uploaded to the database.

6.2.3. Upload

To upload the data, an EC2 instance hosting a simple Windows server was chosen to act as the necessary platform that would handle the database uploading. Rather than rely on the lambda to send data to this server, the server instead downloads data from the S3 'Output-cleaned-alex-2fi23o' itself. Inside the server are a pair of programs: a python script to download the data files from the S3 bucket and a .exe file to upload the downloaded files to the database by means of parsing and context scaffolding. These two programs are placed into the Windows server startup file, causing them to automatically run once the Windows server boots.

Finally, the lambda function was then put on a scheduled timer to check the S3 bucket for present files every night at 3:00 AM Pacific time. If files are present upon inspection, the lambda function simply boots the Windows server EC2, triggering its startup programs to handle the remainder of the uploading process.

6.2.4. Storage

Table 3. Data tables for Obfuscated and Aggregated databases.

ObfuscatedData	**AggregatedData**
Crosswalk_File	Average_Employment_Data
ESD_EQUI	Closure_Rate_Data
ESD_Tax	Cost_of_Labor
DOR_reg_tax	Employment_Hours_Data
DOR_qtr_tax	Establishments
Geo_Keys	Min_wage_Data
Geo_Walk	Revenue_Data
User_Login	

Storing the data involved the use of both PostgreSQL and Amazon RDS. Initially, the database schema was created locally using PostgreSQL and the pgAdmin tool. In pgAdmin, the two databases and their tables mentioned in Section 3.2 were created along with proper login and group roles pertaining to each database to ensure security. Upon completion of the overall schema, the local databases were migrated to AWS through Amazon RDS. In addition, to ensure all team members had access to the database, the pgAdmin server was run in a container hosted in an ECS cluster.

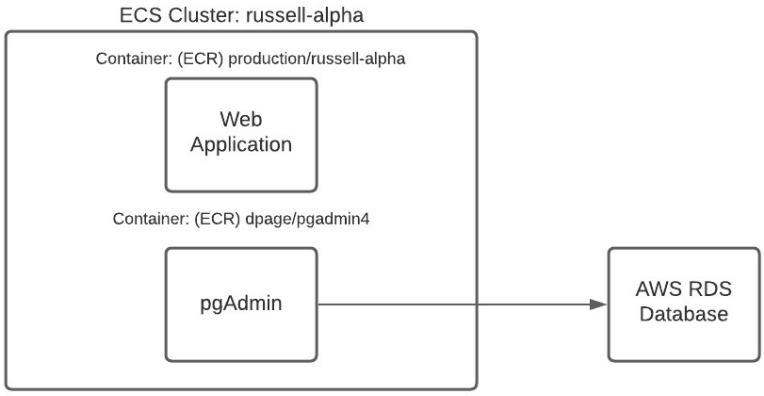
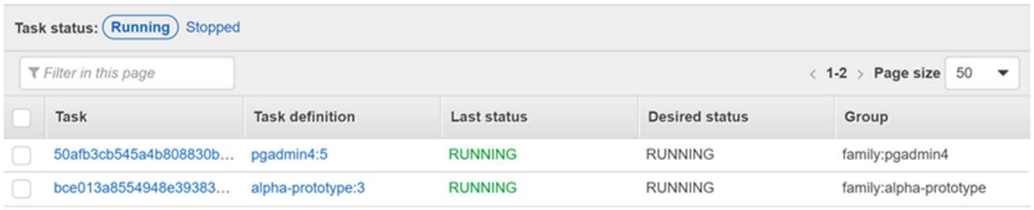


Figure 5. ECS Cluster structure.

Figure 5 shows that the ECS Cluster russell-alpha contains the container that runs the web application, as well as the container that runs the pgAdmin instance, which is connected to the database server. Below is a screenshot that shows the tasks for the ECS Cluster, and the task definitions for pgAdmin and the alpha-prototype web application.



6.3. Delivery

To deliver the data a Web application was made for three different types of users: public, researcher, and admin.

Due to the sensitive nature of the data, these three different types of users come with different permissions.

Table 4. User Permissions.

Permission	Admin	Researcher	Public
Create a user	x		
View ESD data	x	x	
View DOR data	x	x	
View aggregated data	x	x	x

The flow of the website is shown in the flow diagram below.

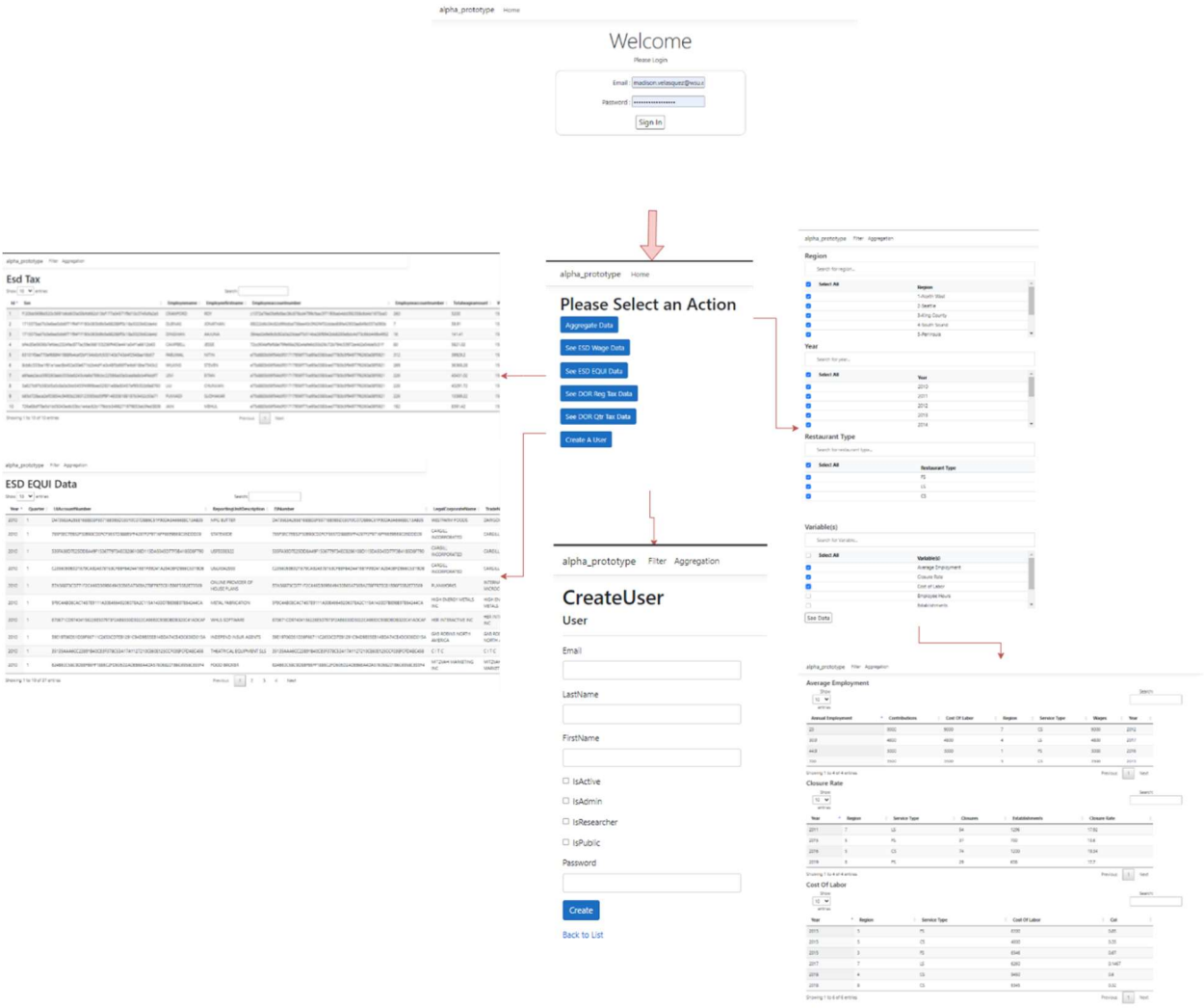


Figure 6. Web application Flow.

6.4. Product Requirement Comparison

Through numerous stakeholder meetings and product requirement gathering, the following requirements were derived:

- Secure data receiving.
- Provable environment security.
- Dataset obfuscation and cleaning.
- Secure data uploading and storage.
- Data delivery through use of Jupyter notebook and aggregations.

From this list, most of the requirements have been completed. The exception holds with the delivery of the aggregated data. The initial plan based off requirements gathering included the

incorporation of an existing Jupyter notebook within a web application to visualize the datasets as the notebook scripts could provide certain aggregations. However, through mentor and team discussion, it was concluded that this was not the most efficient method as the script was incorrectly generating certain aggregations and seemed inefficient in its implementation. Instead, it was suggested that it would be better to exclude the use of the Jupyter notebook and to generate aggregations directly from the database as it created a foundation to incorporate various other data visualization tools for future use.

6.5. Final Prototype Test Results

6.5.1. Receiving

The receiving section of the project was tested using boundary cases such as alternate data types and large file sizes. All file types proved supported by AWS's copy commands and file parsing. Additionally, file sizes proved to be supported up to the upper bound of what the Workspaces instance can handle of 50 gigs. All file types and file sizes were properly disposed of after downloading to the S3.

6.5.2. Processing

6.5.2.1 Obfuscation Lambda Testing

The following section details how the obfuscation lambda function was tested. It focuses on the timing of the function because this was an outcome that could be validated by the administrators. Though the section focuses on the obfuscation lambda, the same techniques can be used to validate the cleaning and uploading lambda which is currently in development.

The timing of the lambda function is broken down into its two dominating components, file size and elements to hash. The values for the slopes of these components were obtained through empirical testing using the projects obfuscation lambda function. To get the test data for the function a .csv file from the ESD was truncated in 20MB chunks to create five different files with identical columns but differing by 20 MB of row data. These five files were fed through the lambda function and the logs were parsed to obtain the input and output timing, hashing timing and max memory usage data. This data was then put into an excel file for analysis, this file is available upon request. The analysis gave the following equation for estimation of time to process a file as a function of file size and number of elements to hash.

Equation 1: $$t = num_{MB} * \left(\frac{1\ second}{11.8\ MB}\right) + num_{elements} * \left(\frac{1\ second}{500000\ elements}\right)$$

With this equation the following plots were created using python and matplotlib.

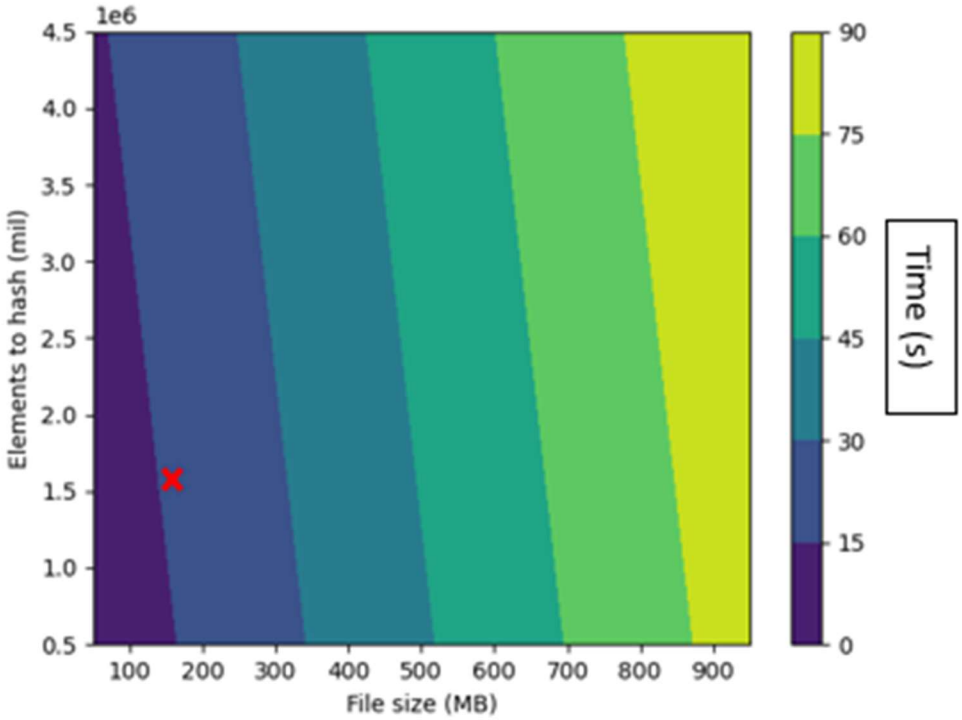


Figure 7. Billable time for obfuscation lambda function.

The red x in figure 7 shows the largest known file to put into the obfuscation lambda. It is slightly to the right of the boundary between the 0-15 color and in the15-30 color, indicating it will take slightly over 15 seconds for the lambda to process the file.

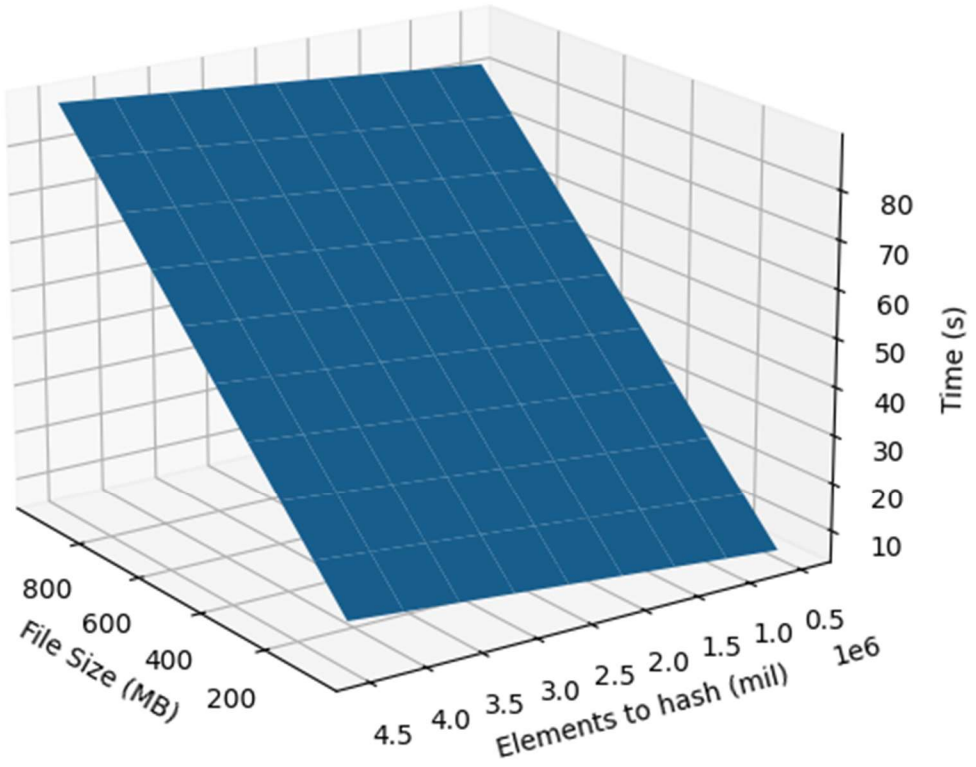


Figure 8. Billable time for obfuscation lambda function.

Figure 8 shows how the time to process a file is dominated by the file size, this is because of input and output costs. The slope of the file size line dominates the slope of the elements to hash line.

This analysis shows that the projects method of obfuscation meets requirements. As the largest file known to the project is 186 MB, and has 1.8 million elements to be hashed, there are no problems with the project's method of obfuscation. With the equation above this gives a reasonable billable time of 19.3 second. This analysis can serve as a basis to estimate and validate the time for any file input into the obfuscation lambda using equation 1.

A bottleneck that could occur using the lambda method, and potentially extend development time, is that lambdas have a hard limit of memory available to them which is 10 gigabytes. The projects lambda's maximum memory usages are proportional to the size of the incoming file. It was seen in the test data that the projects method on average places a coefficient of 5.77 in front of the size of the file, that is a 100 MB incoming file will reach a maximum memory usage of 577 MB (plus or minus a few MB). Therefore, the maximum size file the lambda could handle barring anomalies is 10,000 MB / 5.77, or roughly a 1,732 MB file. So far, the largest file known to the team is 186 MB and the project is well under the file size limit. If there is an unknown file that will enter the system that is larger than 1600 MB, the team recommends implementing either step functions or a method of lambda partitioning that will solve this problem of incoming file size that was discovered during testing.

6.5.3. Storage

To test the accuracy of the database tables and their respective columns, provided sample data was uploaded to each table to ensure each column was storing the proper data and their types. In testing the database, a major obstacle that occurred involved the ESD_EQUI file. As this file required the use of an external dictionary file to translate each column and their respective headers, the team found it to be a challenge as there were discrepancies with the test data and provided dictionaries. It was found that the provided test data was outdated, and a more recent version was required. From there, further cleaning was required to parse the file correctly to match the database types.

6.6. Final Prototype Validation Results

User Stories gathered last semester (taken from the Cpts 421 Technical Specification document):

Administrator User Stories

Story 1: Administrators want to create a secure cloud environment using AWS such that the ESD and DOR will allow the data to be stored in the cloud.

Story 2: Administrators want to reach the goals of security, delivery, and storage within a realistic budget.

Story 3: Administrators want to create an obfuscated dataset in the secure part of the VPC where only admins with authorized IP addresses have access, so it can be delivered to a subnet where assigned researcher IP addresses have access.

Story 4: Administrators want to display the data to the researchers in a helpful way so they can do their research faster.

Story 5: Administrators want to create a website for the researchers so they can do their research in an efficient way.

Researcher User Stories

Story 1: Researchers want to be able to look at the data on a secure website so it can be accessed remotely.

Story 2: Researchers want to have data comprehension tools available on the website to avoid using multiple platforms for one task.

Story 3: Researchers want to access newly uploaded data in a timely fashion to discover new correlations and current trends.

Other Stakeholders User Stories

Story 1: Participants of the Dataset have their data in the dataset and want their information to be secure to protect their identity.

Story 2: Mentors want the data to be secure so the ESD and DOR will continue their partnership with Washington State University.

Story 3: Mentors want to be able to provide their expertise so the team can build a successful project.

Story 4: The ESD and DOR want to deliver sensitive data to a provably secure environment, so the dataset participants' data are safe.

6.6.1. Receiving

User stories met with receiving results:

- **Admin user story 1:**
  - This story was upheld during the receiving phase of the process by using Amazon Workspaces to insulate the data by downloading while already inside the security of AWS.

- **Other stakeholder's user story 1:**
  - This story was upheld by using the insulated Workspaces environment to move the raw data as well as by automatically deleting all traces of the raw data from the Workspaces instance after it has landed in the first S3 bucket.

6.6.2. Processing

User stories met with processing results:

- **Researcher user story 3:**
  - The process from the first S3 to the database uploading is completely automated and highly efficient, allowing the data to be ready in a timely fashion.

- **Other stakeholders' user story 4:**
  - This was met as the data is fully hashed and obfuscated properly through the lambda functions and is processed under full encryption services provided by AWS.
- **Admin user story 3**
  - This was met through the obfuscation lambda residing in the AWS native services cloud. The obfuscation lambda then sends the data to the less secure VPC.
- **Admin user story 2**
  - The project met this story through choosing an architecture and AWS services that optimize for cost, and programming languages and libraries that optimize for speed.

6.6.3. Delivery

User stories met with delivery results:

- **Admin user story 5:**
  - This was met because a beta version of the website was created with a specific role for researchers to view ESD and DOR data efficiently using ajax data tables.

- **Researcher user story 1:**
  - This was accomplished by implementing a secure login system in the website using a database of users and backend with c#. Once the website is deployed it can be accessed from anywhere.

- **Researcher user story 3:**
  - The website grabs data straight from the database every time the user request it on the website, the database is freshly updated with data using lambda functions when data is uploaded.

7. Broader Impacts

A crucial intended impact on another project is the impact this work has for the Carson College of Business at Washington State University. The school has been working on a project for getting this data into a public facing website, with built in tools such as Microsoft BI for data visualization. Our work on this project will be the backbone of the CCB project, and this work will be handed off to them for their future extension. Because of this further extension that will be applied to our work, we created our web application and user tables to allow various user types, allowing for a public user who can only view the aggregated data. We also designed the database to store the obfuscated, raw data from the departments, so data visualizations can be applied to row data as well.

8. Limitations and Recommendations for Future Work

The main functionality of the data pipeline has been achieved, but the resources (workspace, database, ECS containers) are not their production AWS architecture containers. However, these production containers are existing, the team did not want to delete old resources so close to the demo for accidentally deleting a resource that would cause the project to not work for the demo. These resources (workspace, database, ECS containers) need to be configured into the production architecture and the whole systems needs to have a security review. Additional future work in relation to security include deleting unused resources, increasing the percentage for the AWS Foundational Security Best Practices v1.0.0 metric, and creating a conformance pack.

In its current state the project does not perform all the aggregations requested. The process of matching up the aggregations to perfectly meet the desires of the researchers is an ongoing process. This process will undoubtedly require a few more validation and verification meetings with the researchers to ensure the calculations are producing the expected results. A potential source of test data for this can be old files used for and the results obtained from the “Justin” code.

Unfortunately, due to time and resource constraints, work on the Public data visualization website aspect of the project has not begun. As a silver lining it looks as though the Carson College of Business may shift some of their analysis and visualization projects to use Tableau, and this extra time will give space to decide whether to use Microsoft PowerBI or Tableau. With this in mind future work on this could include: determining the software to use for analysis (Tableau, Microsoft PowerBI), figuring out which database works best for the analysis software, if the Aurora database a hurdle, use AWS Glue to duplicate the aurora database to the desired database.

9. Summary/Conclusions

At its final state, the project was a success and resolved most if not all the immediate issues presented to our team. The data is received securely through Amazon Workspaces and downloaded into the S3 landing zone, processed using lambda functions to hash, obfuscate, and clean the data, uploaded to the database using a Windows server EC2 instance, and a secure web application hosted in the cloud is used to pull data from the database. For future teams, the project requires the aggregations to be finished, as more details are needed to complete them. Also, some AWS instances need to be properly scaled for a production level, for example the size capacity on our EC2 servers and the RDS Database. This step could be finalized once more progress is made with the CCB's website project.

We realize that another team in the near future could be taking over this project, and we are open to assist in any way to provide insight, answer questions, provide documents from throughout the project, and any other help necessary. We understand the hurdle of tackling cloud computing for the first time.

10. Acknowledgements

Special thanks to our mentors William Bonner and Geoff Allen, researchers Brad Gaolach and Mark Beattie, our professor Dr. Bolong Zeng, and Washington State University for supporting us in this project.

11. References

1. AWS Documentation: AWS Documentation (amazon.com)
2. Laforet, Marc. Feb 6th, 2019. "Comparing Python and SQL for building Data Pipelines". *Towards Data Science.* Comparing Python and SQL for Building Data Pipelines | by Marc Laforet | Towards Data Science
3. Gilbert H., Handschuh H. (2004) Security Analysis of SHA-256 and Sisters. In: Matsui M., Zuccherato R.J. (eds) Selected Areas in Cryptography. SAC 2003. Lecture Notes in Computer Science, vol 3006. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-24654-1_13

12. Appendix

Last semester's work consisted of thorough documentation and research for designing the product. Described below are the documents created to construct our product design and explain what each document contributes.

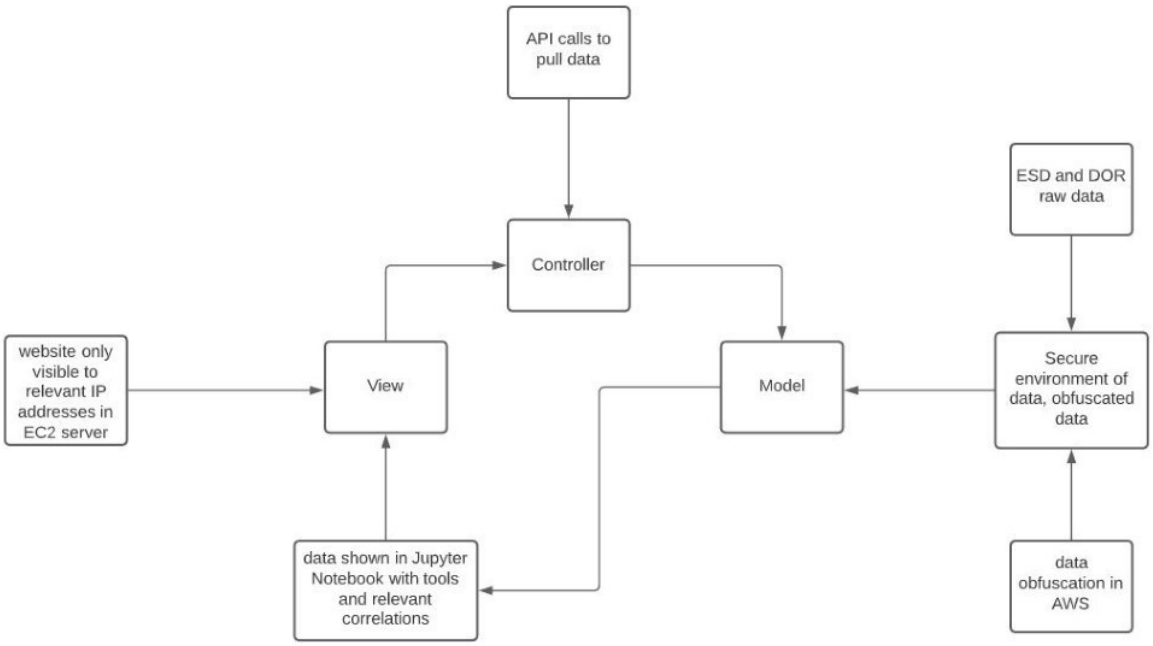
Project Description:

Here we highlighted the complete description and expected scope of the project. In this document the features and limitations were also described along with a list of the stakeholders.

Technical Description:

In this document we identified the stakeholders' requirements and needs. User stories were created and listed to gather requirements for the product. Wire frames for delivering the data on website were also drawn to aid the stakeholders to understand the developer goals. This document also included a preliminary implementation plan for the product.

MVC architecture



Solution Approach:

In this document a concept generation table was presented that included the AWS infrastructure, security, receiving the data, obfuscating the data, storing the data, and delivering the data. The solution for each these sections was described with a solution selection process.

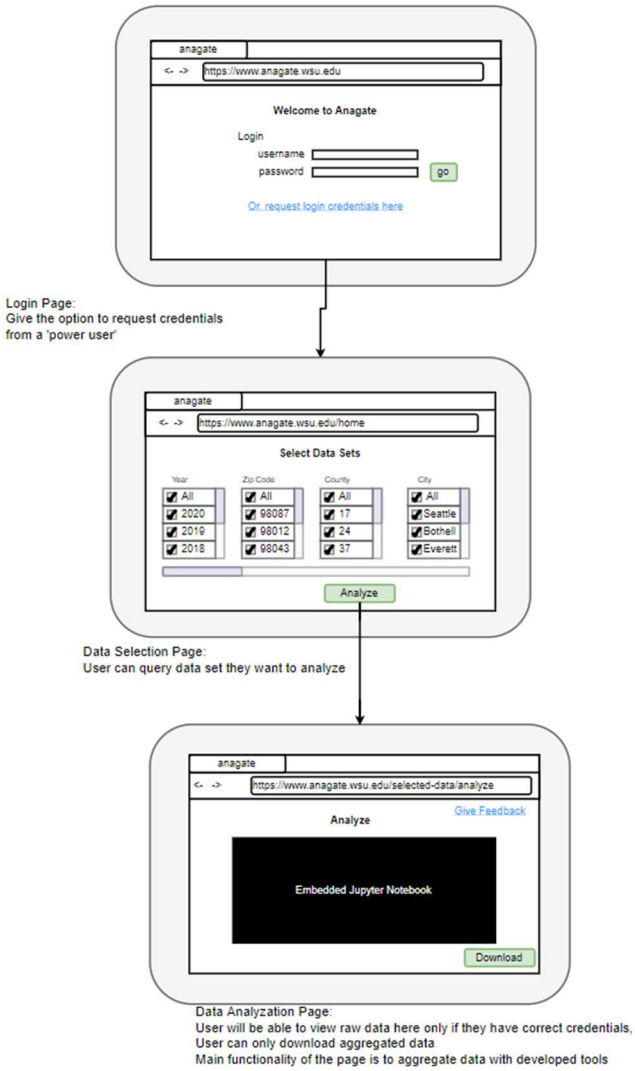
Concept Generation table:

AWS Cloud Infrastructure **Russell**	**Security Alex**		**Receive Data Adam**	**Obfuscate Data Adam**	**Store Data Danny**	**Deliver Data Madison**
	Outside Server	**Inside Server**				
ECS (docker containers & EC2 instances)	AWS Config	Splunk	Online upload interface for data	AWS tools for masking and processing data	PostgreSQL	Web application
EKS	IAM Analyzer	Cortex	Secure email portal	Step Functions	MongoDB	Desktop application
Virtual Machine	AWS Guard Duty	AWS Systems Manager	Port portal to interface to streamline use	Event Bridge	DynamoDB (AWS)	Amazon Workspace
	AWS Security Hub			Lambda functions	Amazon RDS	Jupyter Notebook
	Security Groups				Aurora	
	VPC Flow Logs					
	AWS CloudTrail & CloudWatch					

Alpha Prototype:

In this document the details for the alpha prototype were described for the web application, lambda functions, the process for receiving the data, and the database.

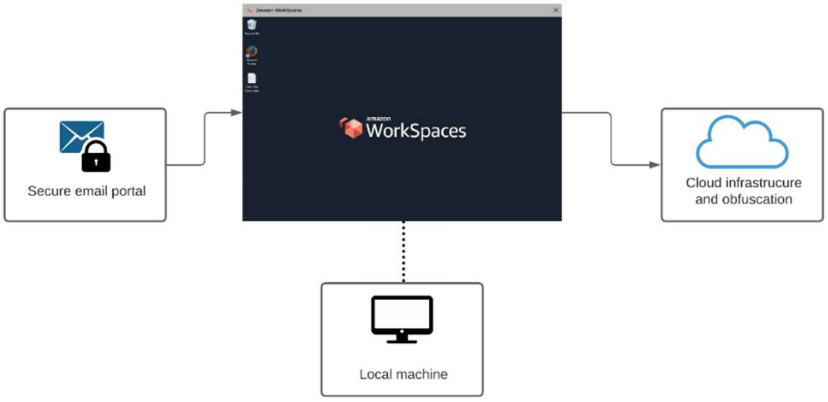
Initial wireframes for web application:



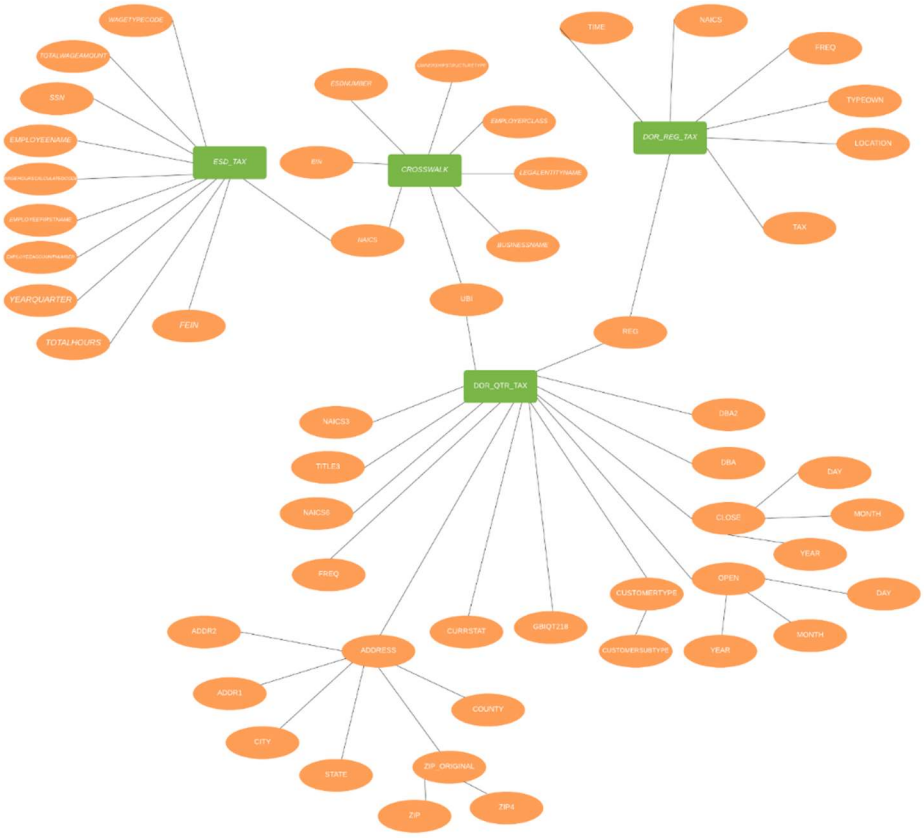
Lambda Functions output:

```
Pandas read in the csv in 2.4721 seconds
Pandas hashed the ESD_UBI_Crosswalk csv in 2.5958 seconds. There were 820,741 rows.
                                      ESDNumber  ... NAICSCode
0  4EB36F1CFE45643B63614FD3C03CDC297BDEF3CAC0A653...  ...  425120.0
1  1775746E642BB602C0FC6A9723A2B6D0B36D7337D8838F...  ...  311511.0
2  279E940546BA6B8A693BC85E7BB7C22EDAD54D18177E5A...  ...  812320.0
```

Receiving data design:



Initial Database Schema:



The semester ended with creating a method to receive the data with amazon workspaces. A code Repository was also created using CodeCommit on AWS, and an outline of the frontend was also written in HTML.